6_RAG

March 4, 2025

Tema 3: Computación Inteligente (LLM)

Generación Aumentada por Recuperación (RAG)

Prof. Wladimir Rodríguez

wladimir@ula.ve

Departamento de Computación

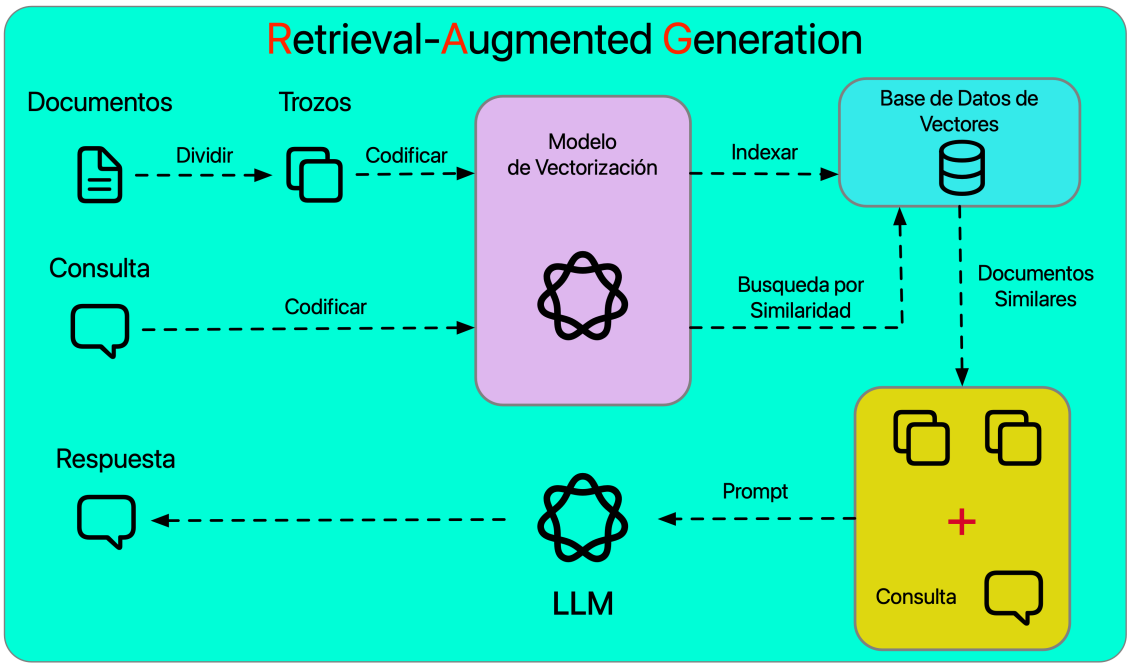
0.1 ¿Qué son los sistemas RAG?

- Combinación de recuperación de información y generación de texto
- Desarrollados para superar limitaciones de los LLM tradicionales
- Permiten a los modelos acceder a conocimiento externo en tiempo real
- Arquitectura híbrida que separa almacenamiento de conocimiento y razonamiento

0.2 El problema que resuelven

- **Conocimiento limitado**: Los LLM solo conocen lo que aprendieron en entrenamiento
- **Alucinaciones**: Generación de información incorrecta pero plausible
- **Ventanas de contexto finitas**: Limitación en la cantidad de texto procesable
- **Opacidad**: Fuentes de información no verificables ni citables

0.3 Arquitectura básica de RAG



0.4 ¿Cómo funciona un sistema RAG?

1. Usuario envía una consulta
2. Sistema convierte consulta en vector (embedding)
3. Búsqueda de información similar en base de conocimiento
4. Recuperación de fragmentos relevantes
5. Modelo recibe consulta original + contexto recuperado
6. Generación de respuesta fundamentada
7. Entrega al usuario con posibles citaciones

0.5 Ventajas de los sistemas RAG

- **Información actualizada**: Acceso a datos recientes
- **Reducción de alucinaciones**: Respuestas basadas en fuentes concretas
- **Transparencia**: Capacidad de citar fuentes
- **Personalización**: Adaptable a dominios específicos
- **Eficiencia**: No requiere reentrenamiento del modelo

0.6 Componentes detallados: Base de conocimiento

- **Tipos de información almacenada**:
  - Documentos textuales (artículos, manuales, libros)
  - Datos estructurados (bases de datos)
  - Código fuente y documentación técnica
  - Contenido web procesado
- **Indexación y procesamiento**:
  - División en fragmentos (“chunking”)

  - Generación de metadatos
  - Creación de embeddings para búsqueda semántica

0.7 Componentes detallados: Sistema de recuperación

- **Métodos de búsqueda**:
  - Similitud coseno entre vectores
  - Algoritmos híbridos (BM25 + semántica)
  - Búsqueda densa de pasajes (DPR)
- **Técnicas avanzadas**:
  - Reordenamiento (re-ranking)
  - Expansión de consultas
  - Filtrado por metadatos (tiempo, categoría, etc.)

0.8 Componentes detallados: Modelo generativo

- **Preparación de prompts**:
  - Estructuración de la consulta
  - Incorporación del contexto recuperado
  - Instrucciones específicas para el modelo
- **Generación optimizada**:
  - Verificación interna de consistencia
  - Detección de contradicciones
  - Citación de fuentes

0.9 RAG vs. Fine-tuning

	Fine-tuning	RAG
Conocimiento	Integrado en pesos del modelo	Separado y accesible
Actualización	Requiere reentrenamiento	Actualización simple de base
Transparencia	Limitada (caja negra)	Alta (fuentes citables)
Recursos	Intensivo en computación	Intensivo en almacenamiento
Personalización	Adaptación profunda al dominio	Flexibilidad en fuentes

0.10 Aplicaciones prácticas

- **Empresariales**:
  - Sistemas de soporte técnico y atención al cliente
  - Asistentes de investigación legal y financiera
  - Gestión interna de conocimiento corporativo
- **Educativas**:
  - Tutores personalizados con acceso a material específico
  - Sistemas de respuesta a preguntas académicas
  - Asistentes de investigación científica

0.11 Aplicaciones prácticas (cont.)

- **Salud**:

  - Asistentes médicos con acceso a literatura científica
  - Sistemas de apoyo al diagnóstico
  - Análisis de historiales clínicos
- **Software**:
  - Asistentes de programación con acceso a documentación
  - Sistemas de gestión de código y documentación
  - Automatización de respuestas técnicas

0.12 Desafíos técnicos

- **Calidad de recuperación**:
  - Relevancia de los resultados
  - Manejo de consultas ambiguas
- **Integración de contexto**:
  - Limitaciones de ventana de contexto
  - Priorización de información
- **Mantenimiento de conocimiento**:
  - Actualización continua
  - Detección de información contradictoria

0.13 Estrategias de implementación

- **Selección de base vectorial**:
  - Opciones: Pinecone, Weaviate, Milvus, Chroma, etc.
  - Consideraciones de escala y rendimiento
- **Modelos de embedding**:
  - OpenAI, Cohere, BERT, Sentence-BERT, E5
  - Balance entre calidad y eficiencia
- **Estrategias de chunking**:
  - Por tamaño fijo vs. semántico
  - Fragmentos jerárquicos vs. solapados

0.14 Evaluación de sistemas RAG

- **Métricas clave**:
  - Precisión de recuperación
  - Calidad de respuestas generadas
  - Velocidad de respuesta
  - Tasa de citación correcta
- **Metodologías**:
  - Evaluación humana
  - Benchmarks automatizados
  - Pruebas A/B con usuarios reales

0.15 Tendencias emergentes

- **RAG multimodal**:
  - Incorporación de imágenes, audio y video
  - Razonamiento entre diferentes formatos

- **RAG con agentes**:
  - Sistemas que deciden autónomamente qué información recuperar
  - Razonamiento multi-paso con búsquedas dinámicas
- **RAG personalizado**:
  - Adaptación a preferencias del usuario
  - Mantenimiento de bases de conocimiento individuales

0.16 Arquitecturas avanzadas

- **RAG reflexivo**:
  - Auto-evaluación de resultados
  - Refinamiento iterativo de consultas
- **RAG híbrido**:
  - Combinación con conocimiento paramétrico
  - Sistemas con múltiples fuentes de recuperación
- **RAG con memoria**:
  - Mantenimiento de contexto conversacional
  - Aprendizaje de interacciones previas

0.17 Consideraciones éticas y de gobernanza

- **Calidad de la información**:
  - Verificación de fuentes
  - Prevención de desinformación
- **Privacidad**:
  - Manejo de datos sensibles
  - Controles de acceso
- **Transparencia**:
  - Explicabilidad de resultados
  - Trazabilidad de fuentes

0.18 Conclusiones

- Los sistemas RAG representan un paradigma nuevo en IA
- Combinan la potencia generativa con precisión informativa
- Ofrecen soluciones a problemas fundamentales de los LLM
- Permiten personalización sin reentrenamiento
- Evolución continua hacia sistemas más sofisticados

0.19 Importar los paquetes necesarios

[1]:
```
import os
import logging
from langchain_community.document_loaders import UnstructuredPDFLoader
from langchain_text_splitters import RecursiveCharacterTextSplitter
from langchain_community.vectorstores import Chroma
from langchain_ollama import OllamaEmbeddings
from langchain.prompts import ChatPromptTemplate, PromptTemplate
```

```
from langchain_ollama import ChatOllama
from langchain_core.output_parsers import StrOutputParser
from langchain_core.runnables import RunnablePassthrough
from langchain.retrievers.multi_query import MultiQueryRetriever
import ollama
```

0.20 Registrar Eventos

[2]:
```
# Configurar registro de eventos (logging)
logging.basicConfig(level=logging.INFO)
```

0.21 Definir Constantes

[3]:
```
# Constantes
CAMINO_DOCUMENTOS = "../datos/IAG.pdf"
NOMBRE_MODELO = "llama3.2"
MODELO_VECTORIZACION = "nomic-embed-text"
NOMBRE_ALMACENAMIENTO_VECTOR = "simple-rag"
```

0.21.1 Función para leer archivos PDF

[4]:
```
def leer_pdf(camino_documentos):
    """cargar documentos."""
    if os.path.exists(camino_documentos):
        cargador = UnstructuredPDFLoader(file_path=camino_documentos)
        data = cargador.load()
        logging.info("PDF cargado exitosamente.")
        return data
    else:
        logging.error(f"archivo PDF no encontrado: {camino_documentos}")
        return None
```

0.21.2 Función para generar trozos de los documentos

[5]:
```
def dividir_documentos(documentos):
    """Dividir documentos en trozos pequeños."""
    text_splitter = RecursiveCharacterTextSplitter(chunk_size=1200,␣
 ↪chunk_overlap=300)
    trozos = text_splitter.split_documents(documentos)
    logging.info("Documentos divididos en trozos.")
    return trozos
```

0.21.3 Función para crear base de datos de vectores

[6]:
```
def crear_bd_vector(trozos):
    """Crear una base de datos de vectores a partir de los trozos del documento.
    """
    # Bajar el modelo de vectorizaciónl si no esta disponible
    ollama.pull(MODELO_VECTORIZACION)

    vector_db = Chroma.from_documents(
        documents=trozos,
        embedding=OllamaEmbeddings(model=MODELO_VECTORIZACION),
        collection_name=NOMBRE_ALMACENAMIENTO_VECTOR,
    )
    logging.info("Vector database created.")
    return vector_db
```

0.21.4 Función para crear el recuperador de fragmentos

[7]:
```
def crear_recuperador(vector_db, llm):
    """Create un recuperador de multi-consulta."""
    QUERY_PROMPT = PromptTemplate(
        input_variables=["question"],
        template="""Eres un asistente de modelo de lenguaje de IA. Tu tarea es␣
    ↪generar cinco
versiones diferentes de la pregunta del usuario dada para recuperar documentos␣
    ↪relevantes de
una base de datos de vectores. Al generar múltiples perspectivas sobre la␣
    ↪pregunta del usuario, tu
objetivo es ayudar al usuario a superar algunas de las limitaciones de la␣
    ↪búsqueda de similitud
basada en la distancia. Proporciona estas preguntas alternativas separadas por␣
    ↪nuevas líneas.
Pregunta original: {question}""",
    )

    recuperador = MultiQueryRetriever.from_llm(
        vector_db.as_retriever(), llm, prompt=QUERY_PROMPT
    )
    logging.info("Crear recuperador.")
    return recuperador
```

0.21.5 Crear cadena de Langchain

[8]:
```
def crear_cadena(recuperador, llm):
    """Crear la cadena"""
    # RAG prompt
```

```
    template = """Responda la pregunta basándose ÚNICAMENTE en el siguiente␣
 ↪contexto:
{context}
Pregunta: {question}
"""

    prompt = ChatPromptTemplate.from_template(template)

    cadena = (
        {"context": recuperador, "question": RunnablePassthrough()}
        | prompt
        | llm
        | StrOutputParser()
    )

    logging.info("Cadena creada exitosamente.")
    return cadena
```

[9]:
```
from pdfminer.utils import open_filename

# Cargar y procesar documento de PDF
data = leer_pdf(CAMINO_DOCUMENTOS)
if data:
    # Dividir los documentos en trozos
    trozos = dividir_documentos(data)

    # Crear la base de datos de vectores
    vector_db = crear_bd_vector(trozos)

    # Iniciálizar el modelo de lenguaje
    llm = ChatOllama(model=NOMBRE_MODELO)

    # Crear el recuperador multi-consulta
    #recuperador = crear_recuperador(vector_db, llm)

    # Crear el recuperador
    recuperador = vector_db.as_retriever()

    # Crear la cadena preservandoosla la sintaxis
    cadena = crear_cadena(recuperador, llm)

    # Consulta ejemplo
    pregunta = "Que es GuIA"

    # Obtener la respuesta
    respuesta = cadena.invoke(input=pregunta)
    print("Respuesta:")
```

```
    print(respuesta)
else:
    print('Archivo no encontrado')
```

```
INFO:pikepdf._core:pikepdf C++ to Python logger bridge initialized
INFO:root:PDF cargado exitosamente.
INFO:root:Documentos divididos en trozos.
INFO:httpx:HTTP Request: POST http://127.0.0.1:11434/api/pull "HTTP/1.1 200 OK"
INFO:chromadb.telemetry.product.posthog:Anonymized telemetry enabled. See
https://docs.trychroma.com/telemetry for more information.
INFO:httpx:HTTP Request: POST http://127.0.0.1:11434/api/embed "HTTP/1.1 200 OK"
INFO:root:Vector database created.
INFO:root:Cadena creada exitosamente.
INFO:httpx:HTTP Request: POST http://127.0.0.1:11434/api/embed "HTTP/1.1 200 OK"
INFO:httpx:HTTP Request: POST http://127.0.0.1:11434/api/chat "HTTP/1.1 200 OK"

Respuesta:
Según el texto proporcionado, GuIA (Guía de buenas prácticas) es una iniciativa
que tiene como objetivo generar un ecosistema para compartir y conocer las
mejores prácticas en inteligencia artificial ética y normativa. Su objetivo es
crear comunidad alrededor de las buenas prácticas aterrizadas en inteligencia
artificial ética y normativa, con tres pilares estratégicos: investigación,
divulgación y formación.
```

[10]:
```
# Consulta ejemplo
pregunta = "cuales son las buenas prácticas"

# Obtener la respuesta
respuesta = cadena.invoke(input=pregunta)
print("Respuesta:")
print(respuesta)
```

```
INFO:httpx:HTTP Request: POST http://127.0.0.1:11434/api/embed "HTTP/1.1 200 OK"
INFO:httpx:HTTP Request: POST http://127.0.0.1:11434/api/chat "HTTP/1.1 200 OK"

Respuesta:
Las buenas prácticas recomendadas en el documento se pueden resumir de la
siguiente manera:

1. **Equidad**: Articular marcos que equilibren objetivos y diferentes
definiciones de equidad, priorizando la equidad en algunas situaciones
hipotéticas comunes.
2. **Protección de la privacidad**: Encontrar un enfoque que equilibre
adecuadamente la privacidad y la utilidad para la tarea en cuestión. Utilizar
técnicas como "PATE" (Private Aggregation of Teacher Ensembles) para transferir
conocimiento de modelos maestros a modelos estudiantes con una privacidad
inmediata.
3. **Seguridad**: Sigue los procesos de mejores prácticas establecidos para el
software criptográfico y crítico, como:
```

```
 * Uso de enfoques basados en principios y ser demostrables.
 * Publicación de nuevas ideas revisadas y aprobadas por la comunidad.
 * Código abierto de componentes de software críticos.
 * Contratación de expertos para su revisión en todas las etapas de diseño y
desarrollo.
4. **Investigación sobre la privacidad**: Realizar investigaciones sobre la
privacidad con casos de uso, como evaluar conjuntos de datos de entrenamiento
para evitar posibles fuentes de sesgo y crear modelos de entrenamiento para
eliminar o corregir dichos sesgos.

Es importante mencionar que estas buenas prácticas se enfocan en la inteligencia
artificial ética y su aplicación en diferentes sectores, como la privacidad, la
equidad y la seguridad.
```

[11]:
```python
# Consulta ejemplo
pregunta = "que es la SEDIA"

# Obtener la respuesta
respuesta = cadena.invoke(input=pregunta)
print("Respuesta:")
print(respuesta)
```

```
INFO:httpx:HTTP Request: POST http://127.0.0.1:11434/api/embed "HTTP/1.1 200 OK"
INFO:httpx:HTTP Request: POST http://127.0.0.1:11434/api/chat "HTTP/1.1 200 OK"

Respuesta:
La SEDIA se refiere a la Secretaría de Estado de Digitalización e Inteligencia
Artificial.
```

[12]:
```python
# Consulta ejemplo
pregunta = "Ejemplo de buenas practicas"

# Obtener la respuesta
respuesta = cadena.invoke(input=pregunta)
print("Respuesta:")
print(respuesta)
```

```
INFO:httpx:HTTP Request: POST http://127.0.0.1:11434/api/embed "HTTP/1.1 200 OK"
INFO:httpx:HTTP Request: POST http://127.0.0.1:11434/api/chat "HTTP/1.1 200 OK"

Respuesta:
Según el contexto proporcionado, un ejemplo de buena práctica en el uso de
inteligencia artificial ética es la evaluación de los conjuntos de datos de
entrenamiento para evitar posibles fuentes de sesgo. Esto se menciona en el
documento como una práctica recomendada para abordar la equidad en la IA.
```

[ ]:
```python

```